

C

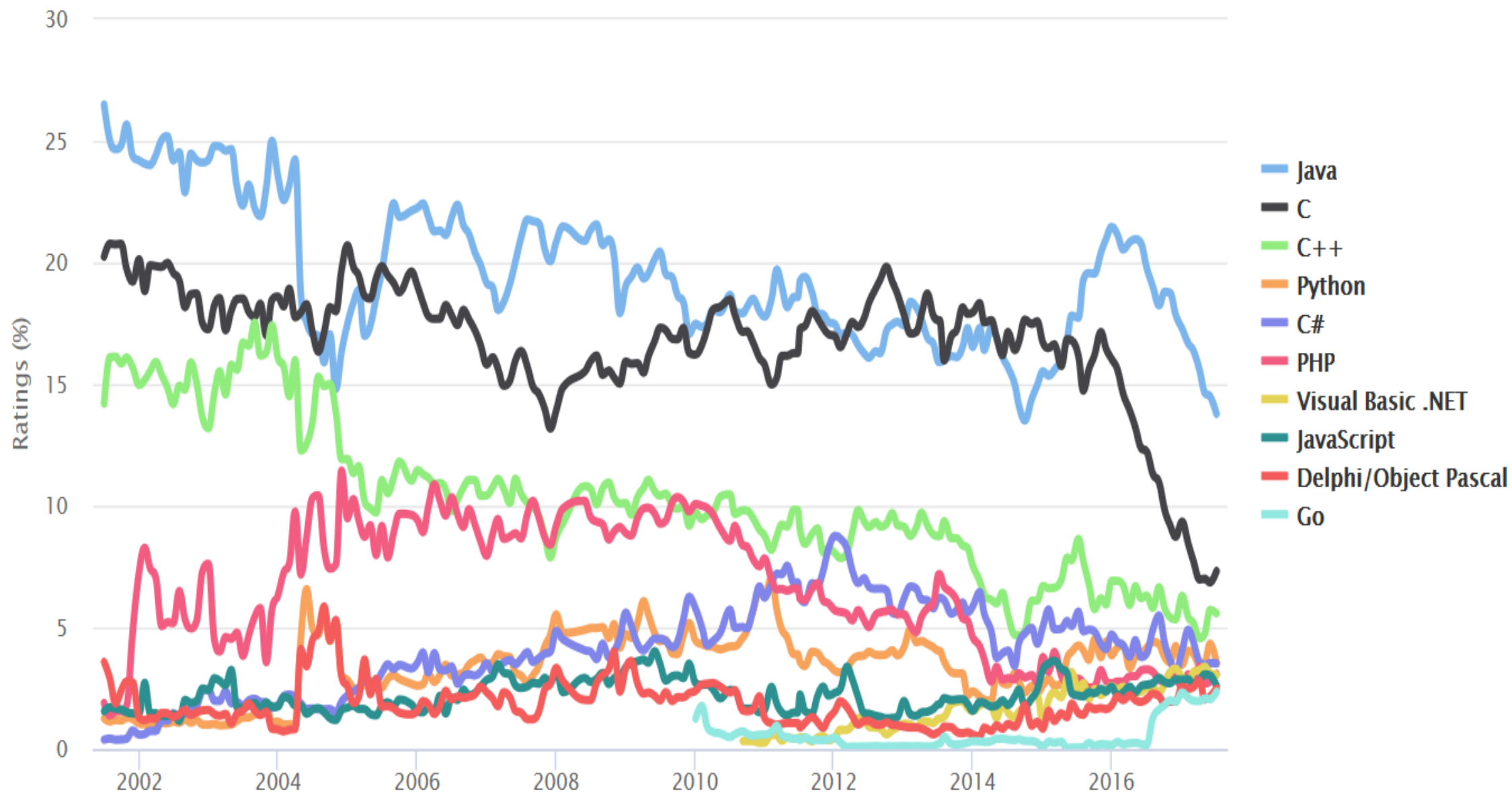
C가 뭐죠?

- C는 1972년 켄 톰슨과 데니스 리치가 벨 연구소에서 일할 당시 새로 개발된 유닉스 운영 체제에서 사용하기 위해 개발한 **프로그래밍 언어**이다. (출처: 위키피디아)
- 프로그래밍 언어는 C만 있나?
- 프로그래밍언어 순위(2017년 5월)

Jul-17	Jul-16	Programming Language	Ratings
1	1	Java	13.77%
2	2	C	7.32%
3	3	C++	5.58%
4	4	Python	3.54%
5	5	C#	3.52%
6	6	PHP	3.09%
7	8	Visual Basic .NET	3.05%
8	7	JavaScript	2.61%
9	12	Delphi/Object Pascal	2.49%
10	55	Go	2.36%
11	9	Perl	2.33%
12	14	Swift	2.25%
13	11	Ruby	2.25%
14	10	Assembly language	2.24%
15	17	R	2.11%
16	13	Visual Basic	2.10%
17	16	MATLAB	2.01%
18	15	Objective-C	1.90%
19	21	Scratch	1.81%
20	18	PL/SQL	1.55%

TIOBE Programming Community Index

Source: www.tiobe.com





C언어를 왜 배워야 하나요



전체

동영상

이미지

지도

뉴스

더보기

설정

도구

검색결과 약 106,000개 (0.40초)

대학에서 C 언어를 배우는 이유? : 네이버 블로그 - Blog Naver

blog.naver.com/PostView.nhn?blogId=cert1970&logNo=220911298547 ▼

2017. 1. 14. - 가장 평범한 이름의 컴퓨터공학과부터 시작해서 컴퓨터과학과, 정보컴퓨터공학과, 소프트웨어공학과 등 컴퓨터 관련 학과들은 항상 공통으로 ...

자기계발을 멈추면 죽는다 :: C언어를 배워야하는 이유

skmagic.tistory.com > 프로그래밍 > C언어 ▼

2011. 1. 8. - 출처:대용이의 Semantic Widgets 삽질 상자 ㅋ 왜 씨언어를 배워야 할까? 일단 각 대학에서 처음에 씨언어를 가르키는 이유를 말하자면 **C언어**가 ...

C언어 꼭 배워야 하나요? - C/C++ - 나프다Q&A(beta)

qna.iamprogrammer.io > 개발 > C/C++ ▼

2017. 2. 8. - 제가 여기서 궁금한 점은 꼭 C와 C++을 배워야 할까요?(너무 바보 같은 질문 일 수도 ... **C언어**와 C++언어가 언어의 기초이며 운영체제를 구축한 언...

프로그래밍을 배워야 하는 이유는? - YouTube



<https://www.youtube.com/watch?v=SESuctdE9vM> ▼

2013. 4. 3. - 업로더: InShin Hwang

사고하는 법을 배워야 하기 때문이다. ... 한다면 스크래치같은걸로 코딩을 먼저 시작하기보다, 파이썬이나 **C**로 그 자체가 프로그래밍 언어인 것들.

C언어 꼭 배워야 하나요?

■ 개발 ■ C/C++



warmgreen77

1 2월 8

안녕하세요! 저는 이제 막 컴퓨터 세상에 관심을 갖게 된 사람입니다.

유니티 강좌를 듣게 되면서 제대로된 코딩 수업을 처음 듣게 되었습니다.

유니티 엔진 사용법과 C#을 제 첫 언어로 배웠는데요. 너무 재미있더라구요!

제가 여기서 궁금한 점은 꼭 C와 C++을 배워야 할까요?(너무 바보 같은 질문 일 수도 있으나...)

C언어와 C++언어가 언어의 기초이며 운영체제를 구축한 언어라는 것은 알고 있으나,

C#에 비해 C언어와 C++은 무척 어렵고, 그러다 보니 재미가 없습니다. ππ

저는 C언어와 C++을 배워야하는 이유와 재밌게 배우려면 어떤 엔진이나 툴을 같이 배워야 하는지 무척 궁금합니다.

고수님들의 친절한 답변 기다리고 있겠습니다. 감사합니다.



serithemage

2월 8

모든 개발자에게 필수인가? 라는 질문에 저는 글썄요 라고 생각합니다.
필요하다고 생각되면 그때 배워도 됩니다.

배워야 할 이유: 아직은 메이저 언어임, 컴퓨터의 저수준 동작, 특히 OS를 이해하는데 많은 도움이 됨
배우지 말아야 할 이유: 소프트웨어 공학이 발전해 나감에 따라 대체할 수 있는 언어들이 등장하고 있음

2 좋아요



필요하다고 생각되면 그 때 배워도 됩니다.

프로그래밍 공부할 수 있는 사이트

- Google에 물어보니...
 - KISTI 교육센터 (webedu.ksc.re.kr)
 - Codecademy
 - Udacity
 - Khan Academy
 - Coursera
 - edX
 - iTunes University
 - ...

안녕, 세상아!

```
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
```



```
#include <stdio.h>
```

- stdio.h (헤더)파일을 include 합니다.
- 왜?
 - stdio ← standard input output
 - stdio.h 헤더파일 없으면 입출력 X

대표적 헤더파일(Header File)

- math.h – 수학관련 함수 정의
- stdlib.h – 표준라이브러리, 난수생성, 메모리 할당...
- time.h – 시간 관련 함수 정의
- string.h – 문자열 관련 함수 정의

`int main(void)`

- C언어는 main만 바라봐! (가장 먼저 실행되는 함수)
- 출력(int)과 입력(void)이 있는 main 함수

- 자료형

char	문자형	1바이트
int	정수형	4바이트
float	실수형	4바이트
double	실수형	8바이트

```
printf("Hello, World! \n");
```

- Printf() – 출력 함수, () 내용을 출력
- 따옴표("") 사이 문자를 출력
- \n 은 enter 역할
- 하나의 명령이 끝나면 무조건 ; 으로 끝났음을 알려줘야 함

변수

- 변할 수 있는 수
- 변수에는 수, 문자, 포인터 등이 저장됨

```
int i;
```

```
i = 0;
```

```
double j;
```

```
j = 0.0;
```

1. 변수의 이름은 알파벳, 숫자, 언더바(_)로만 구성
2. 대소문자를 구분 (number와 Number은 다른 변수임)
3. 변수 이름의 첫글자는 숫자로 시작할 수 없음
4. 변수 이름에 공백을 포함할 수 없음
5. 예약어(키워드)를 사용할 수 없음(int, main 등)

자료형 전환

- 서로 다른 자료형 간 연산 원칙적으로 불가
(4 + 2.5)
- 가능하지만, 오류 발생 가능
- 변수 앞에 자료형 표기해서 사용

```
int i;  
double j;  
i+j ?
```

```
int i;  
double j;  
i+(int)j
```

연산자

연산자	기능
=	오른쪽의 값을 왼쪽에 대입
+	왼쪽과 오른쪽의 합
-	왼쪽에서 오른쪽을 뺌
*	왼쪽과 오른쪽의 곱
/	왼쪽을 오른쪽으로 나눈 몫
%	왼쪽을 오른쪽으로 나눈 나머지

출력 - printf()

- 출력 서식문자 - 정수출력 %d, 실수출력 %f

```
int num1 = 12345;  
printf("%d", num1);
```

```
double num2 = 3.1415;  
printf("%lf", num2);
```

```
int num1 = 12345;  
double num2 = 3.1415;  
printf("%d %lf", num1, num2);
```


입력 – scanf()

- scanf("서식 문자열", &변수 [,&변수,...]);
- 서식 문자열

서식 문자	내용
%d	정수
%f	실수(float)
%lf	실수(double)
%c	문자
%s	문자열

숫자 입력

```
#include <stdio.h>

int main()
{
    int a;
    scanf("%d", &a);
    printf("%d", a);
}
```

```
#include <stdio.h>

int main()
{
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d, %d", a, b);
}
```

문자 입력

```
#include <stdio.h>

int main()
{
    char a;

    printf("Input : ");
    scanf("%c", &a);

    printf("Output : %c\n", a);
}
```

문자열 출력

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char string[128];
```

```
    scanf("%s", &str[0]);
```

```
    printf("Output : %s\n", string);
```

```
}
```

printf, scanf 연습

- 정수 A와 B를 입력 받아 $A*B$ 값을 출력하시오

조건문

- if, case 등등
- 특정 조건일 경우에 코드를 실행하는 문법

```
if (조건)
{
    **참**일 때 실행되는 명령들
}
```

홀수/짝수 확인 코드

```
int num;
printf("정수를 입력하세요: ");
scanf("%d", &num);
if (num % 2 == 0)
{
    printf("%d는 짝수입니다.\n", num);
}
```

논리 연산자

연산자	의미
	or 연산자
&&	and 연산자

and 연산자

좌	연산자	우	결과
참	and	참	참
참	and	거짓	거짓
거짓	and	참	거짓
거짓	and	거짓	거짓

or 연산자

좌	연산자	우	결과
참	or	참	참
참	or	거짓	참
거짓	or	참	참
거짓	or	거짓	거짓

```
i = 0 || j = 0
```

```
i = 0 && j = 0
```



```
if (num % 2 == 0)
```

- 관계 연산자

연산자	의미
==	좌우가 같으면 참, 다르면 거짓
>	좌측이 더 크면 참, 작거나 같으면 거짓
<	우측이 더 크면 참, 작거나 같으면 거짓
>=	좌측이 더 크거나 같으면 참, 더 작으면 거짓
<=	우측이 더 크거나 같으면 참, 더 작으면 거짓
!=	좌측과 우측이 다르면 참, 같으면 거짓

if – else

```
int num;
printf("정수를 입력하세요: ");
scanf("%d", &num);
if (num % 2 == 0)
{
    printf("%d는 짝수입니다!\n", num);
}
else
{
    printf("%d는 홀수입니다!\n", num);
}
```

else – if

```
int num;
printf("자연수를 입력하세요: ");
scanf("%d", &num);
if (num % 2 == 0)
{
    printf("%d는 2의 배수입니다!\n", num);
}
else if (num % 3 == 0)
{
    printf("%d는 3의 배수입니다!\n", num);
}
else
{
    printf("%d는 2의 배수도, 3의 배수도 아닙니다!\n", num);
}
```

반복문

- 코드를 반복해서 실행하고 싶을 때 사용
- for, while
- 1부터 10까지 정수 출력 코딩

```
#include <stdio.h>

int main(void)
{
    int num = 1;

    while(num <= 10)
    {
        printf("%d\n", num);
        num = num + 1;
    }
}
```

```
#include <stdio.h>

int main(void)
{
    int num = 1;

    for(num = 1; num <= 10;
num = num + 1)
    {
        printf("%d\n", num);
    }
}
```

증감연산자

변수 i 의 값을 a 만큼 증가시키려면

```
i = i + a;
```



```
i += a;
```

연산자	기능
+=	$i = i + a$
-=	$i = i - a$
*=	$i = i * a$
/=	$i = i / a$
%=	$i = i \% a$

증감연산자

- 변수 i 의 값이 1씩 증가 또는 1씩 감소하는 경우

`i = i + 1;`  `i++;`

연산자	기능
<code>i++</code>	<code>i = i + 1</code>
<code>i--</code>	<code>i = i - 1</code>

- `i++`, `++i` 다른 점

`i++`는 i 값을 사용한 뒤에 i 값을 증가시킴

`++i`는 i 값을 변경한 뒤에 i 값을 사용함

while 문

- 조건이 참인 동안 계속 실행
- 1부터 10까지 더하는 코딩

```
while (조건)
{
조건이 참일 때 실행 될 내용
}
```

```
int i = 1;
int sum = 0;

while (i <= 10)
{
    sum = sum + i;
    i = i + 1;
}

printf("%d", sum);
```


while 문 무한루프 오류

변수가 일정

```
int i = 1;

while (i <= 10)
{
    sum += i;
}
```

초기 조건이 거짓

```
int i = 20;

while (i <= 10)
{
    sum += i++;
}
```

for 문

- 조건이 참인 동안 계속 실행 (while과 동일)
- 1부터 10까지 더하는 코딩

```
for (초기화; 조건; 증감)
{
    조건이 참일 때 실행 될 내용
}
```

```
int i;
int sum = 0;

for (i = 1; i <= 10; i++)
{
    sum += i;
}
printf ("%d\n", sum);
```

for, while 연습

- 입력받은 수 평균 출력
- 구구단 출력

함수

- 특정 용도의 코드를 한 곳에 모아 놓은 것

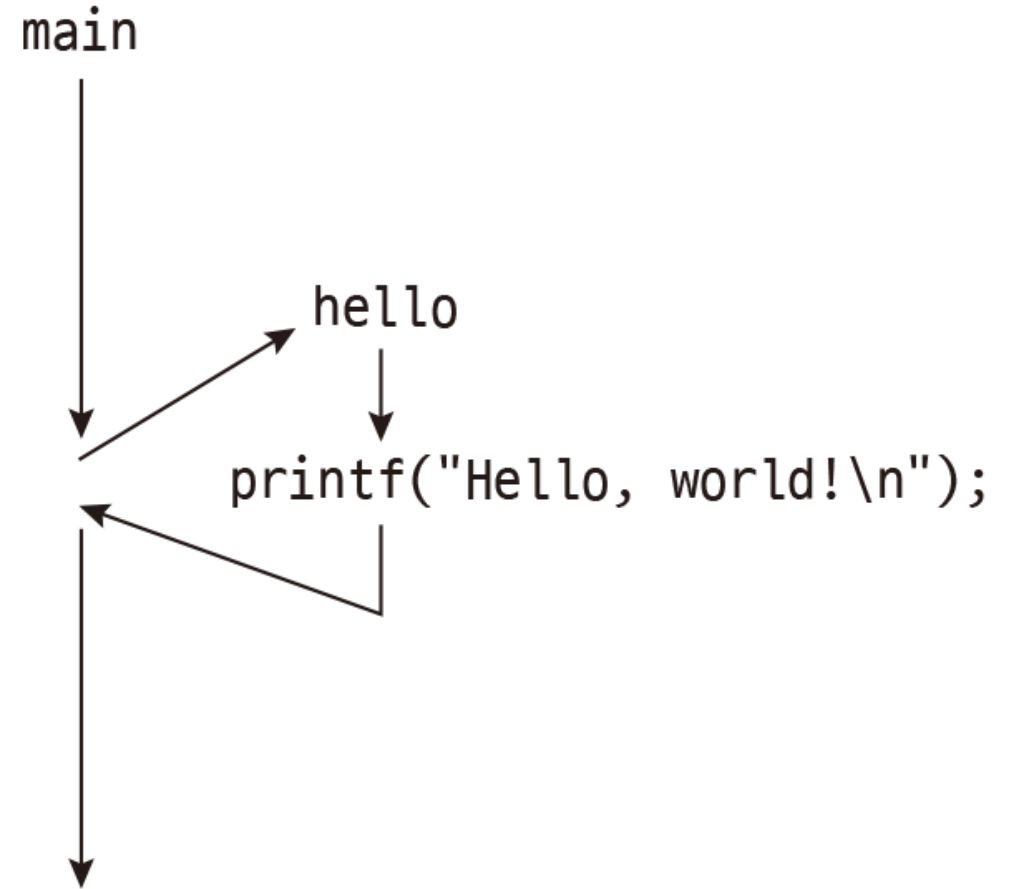
```
#include <stdio.h>
```

```
void hello()  
{  
    printf("Hello, world!\n");  
}
```

```
int main()  
{  
    hello();  
    return 0;  
}
```

반환값자료형 함수이름 (매개변수)
{
코드
return 출력;
}

```
void hello() ③  
{  
    printf("Hello, world!\n"); ④  
} ⑤  
  
int main() ①  
{  
    hello(); ②  
  
    return 0; ⑥  
}
```

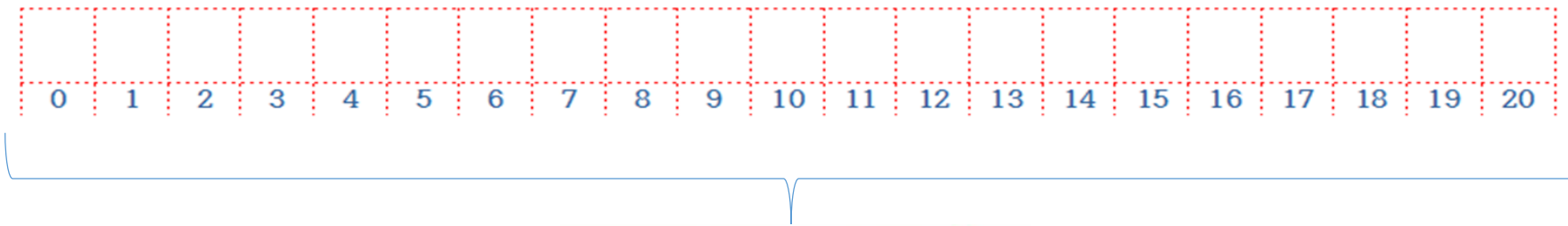


포인터

- 주소를 가지고 있는 변수
- 택배 아저씨가 하는 일
 - 고객의 **주소를 저장**하고 있다가 해당 **주소로 물건을 전달**하는 일
 - 고객의 **주소를 저장**하고 있다가 해당 **주소로 물건을 받아**가는 일
 - '우리에게 '간접 접근' 서비스를 제공한다.'
- 컴퓨팅 세계에서 택배 아저씨와 같은 일을 하는 변수
- 포인터 변수 (포인터라고도 부름)
 - 메모리의 **주소를 저장**하고 있다가 해당 **주소로 데이터를 전달** 하는 일
 - 메모리의 **주소를 저장**하고 있다가 해당 **주소로 데이터를 참조** 하는 일

메모리 구조

- 변수는 메모리에 저장된다.
- 메모리는 바이트 단위로 액세스된다.
 - 첫번째 바이트의 주소는 0, 두번째 바이트는 1,...



메모리와 변수

- `int num1 = 10;`
- 변수는 어디에 생기는가?

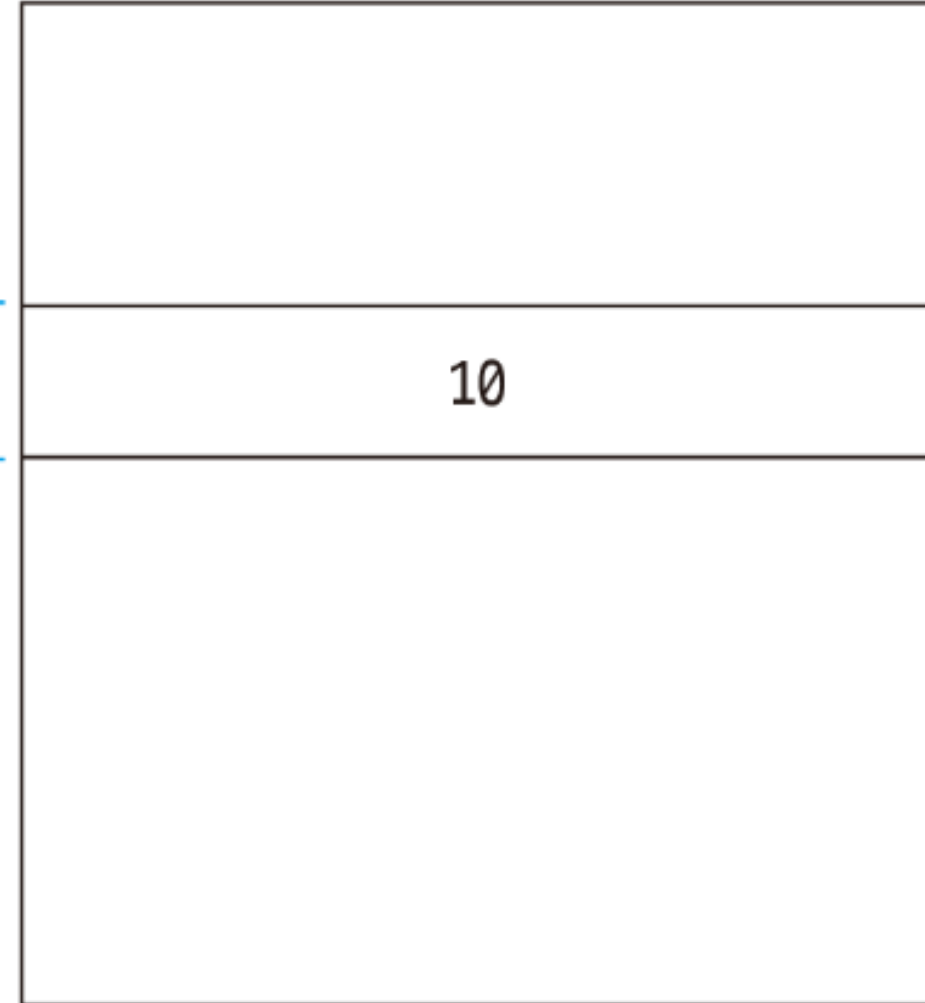
```
#include <stdio.h>
```

```
int main()  
{  
    int num1 = 10;  
    printf("%p\n", &num1);  
    return 0;  
}
```

변수 num1



메모리



메모리

00000000

변수 num1의 메모리 주소 → 008AF7FC

10

FFFFFFFF

포인터 변수 선언

- 메모리 주소는 포인터 (pointer) 변수에 저장

```
자료형 *포인터이름;  
포인터 = &변수;
```

```
int* numPtr  
int * numPtr  
int *numPtr
```

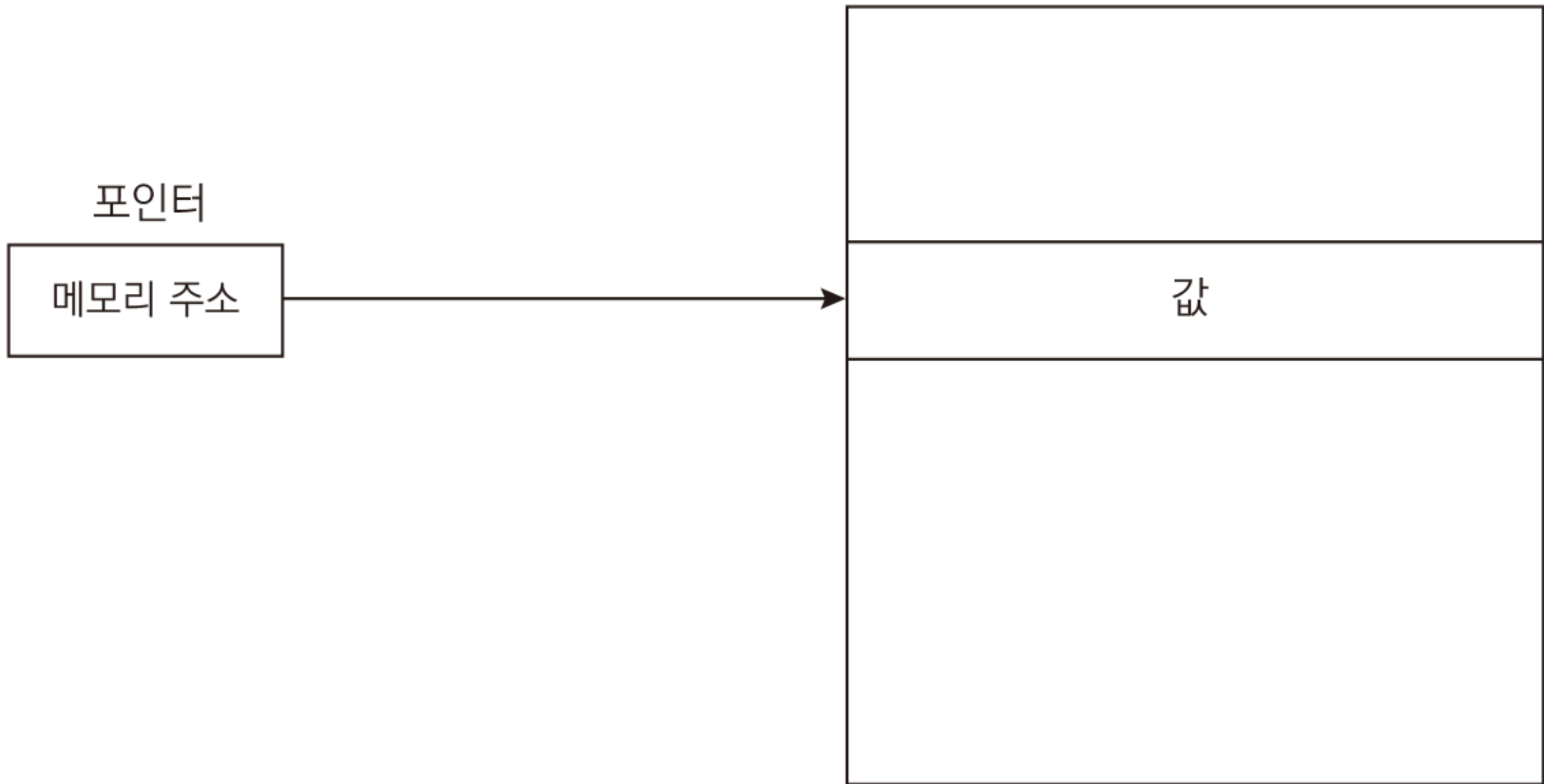
```
#include <stdio.h>  
int main() {  
    int *numPtr;  
    int num1 = 10;  
    numPtr = &num1;  
    printf("%p\n", numPtr);  
    printf("%p\n", &num1);  
    return 0;  
}
```

메모리

포인터

메모리 주소

값



```
int *numPtr  
int num1 = 10;  
  
numPtr = &num1
```



역참조 연산자

*포인터;

- 메모리 주소가 있는 곳으로 이동해서 값을 가져오고 싶다면
역참조 (dereference) 연산자 *를 사용

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int *numPtr;
```

```
    int num1 = 10;
```

```
    numPtr = &num1;
```

```
    printf("%d\n", *numPtr);
```

```
    return 0;
```

```
}
```

역참조 연산자

- 역참조 연산자 *는 포인터 앞에 붙임
- `numPtr` 앞에 *를 붙이면 `numPtr`에 저장된 메모리 주소로 가서 값을 가져옵니다. `numPtr`이 `num1`의 메모리 주소를 저장하고 있으므로 `num1`의 값인 10이 출력됨

포인터는 변수의 주소만 카리킴



역참조는 주소에 접근하여 값을 가져옴



포인터 이용 변수 값 변경

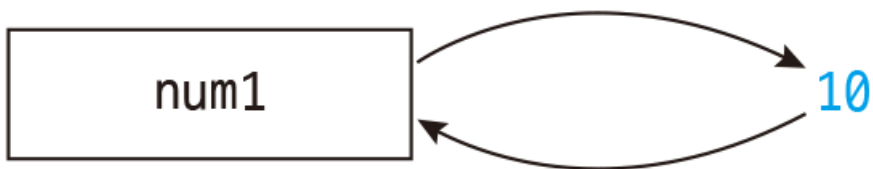
```
#include <stdio.h>

int main()
{
    int *numPtr;
    int num1 = 10;
    numPtr = &num1;
    *numPtr = 20;
    printf("%d\n", *numPtr);
    printf("%d\n", num1);
    return 0;
}
```

역참조로 주소에 접근하여 값을 저장



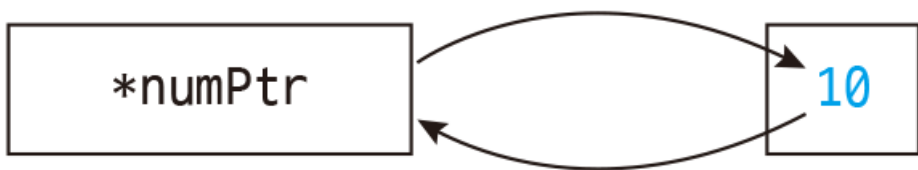
정리



변수는 메모리 주소를 몰라도
값을 가져오거나 저장할 수 있다.



주소 연산자(&)는 변수의 메모리 주소를
구한다.



역참조 연산자(*)는 메모리에 저장된 값에 접근할 수 있다.
즉, 메모리 주소에 접근하여 값을 가져오고 저장한다.



포인터는 변수의
메모리 주소만 가리킨다.

다양한 자료형 포인터 선언

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    long long *numPtr1;
```

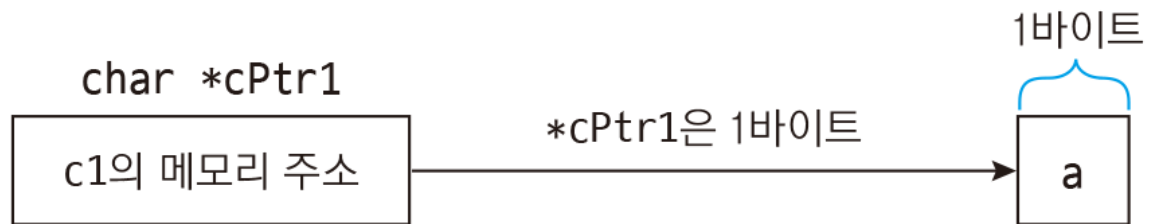
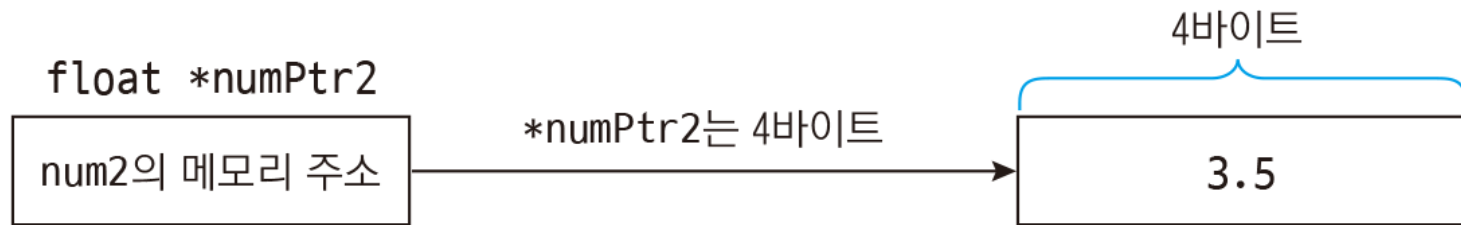
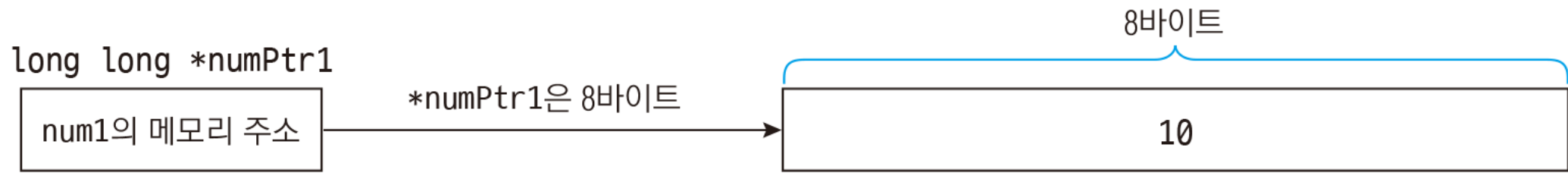
```
    float *numPtr2;
```

```
    char *cPtr1;
```

```
    long long num1 = 10;
```

```
    float num2 = 3.5f;
```

```
    char c1 = 'a';
```



이중포인터

자료형 ****포인터이름;**

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int *numPtr1;
```

```
    int **numPtr2;
```

```
    int num1 = 10;
```

```
    numPtr1 = &num1;
```

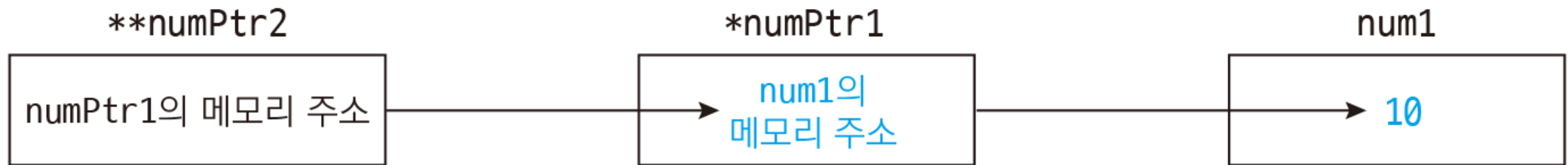
```
    numPtr2 = &numPtr1;
```

```
    printf("%d\n", **numPtr2);
```

```
    return 0;
```

```
}
```

역참조 연산자를 두 번 사용

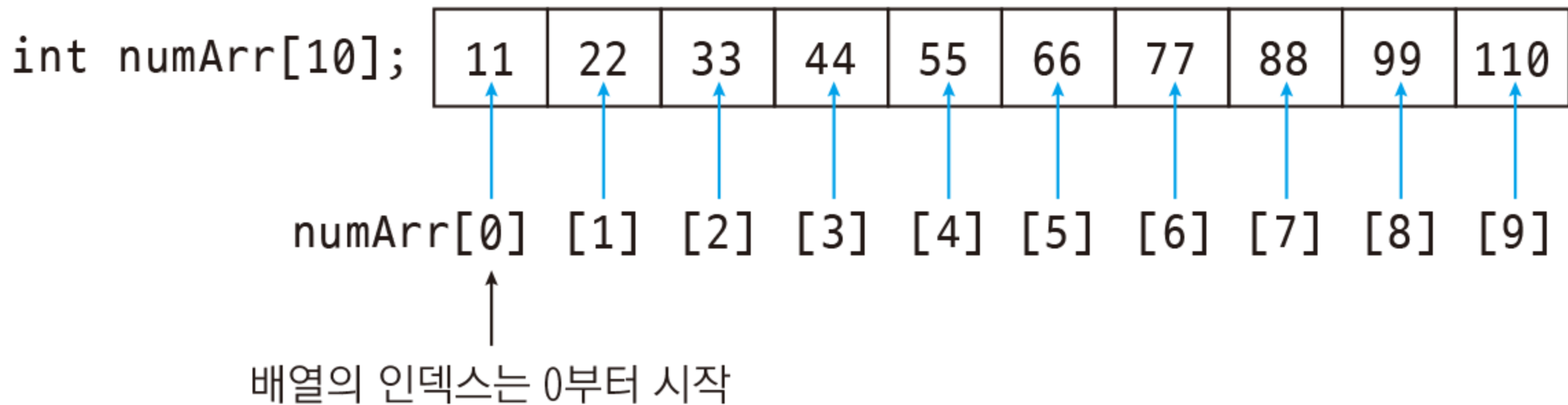


배열 - 배열의 선언

자료형 배열이름[크기];
자료형 배열이름[크기] = { 값, 값, 값 };

- 같은 형태의 자료형이 많을 때 사용하면 효과적
- 배열은 배열명과 변수의 개수, 변수의 자료형으로 선언

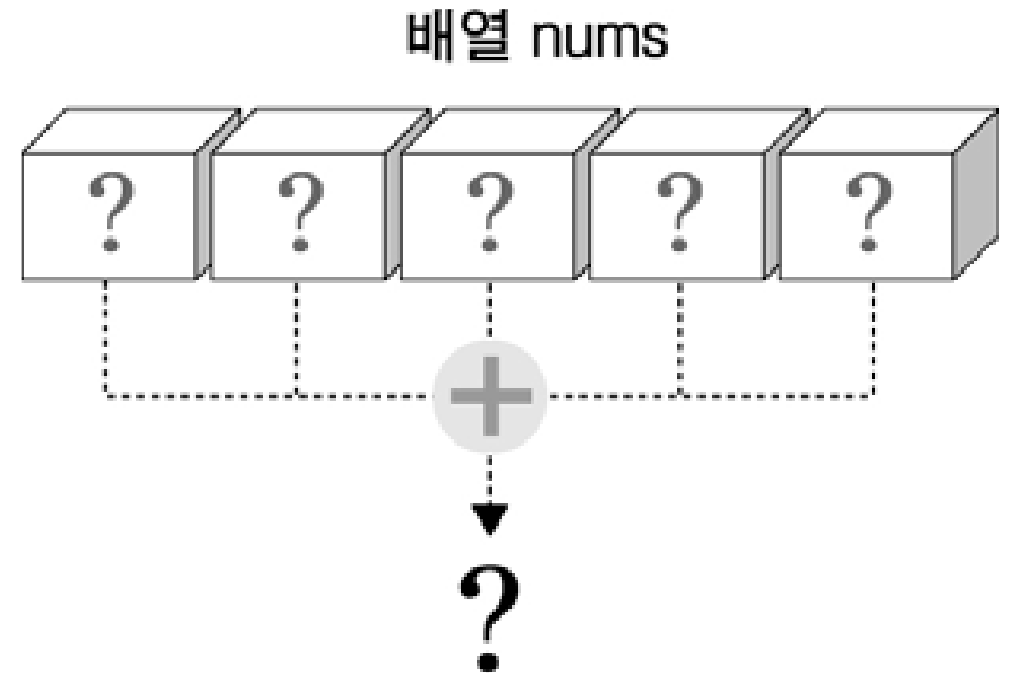
```
#include <stdio.h>
int main()
{
    int numArr[10] = {11, 22, 33, 44, 55, 66, 77, 88, 99, 110};
    printf("%d\n", numArr[0]);
    printf("%d\n", numArr[5]);
    printf("%d\n", numArr[9]);
    return 0;
}
```



배열 초기화

- 배열을 선언하면 처음에 쓰레기 값이 존재

```
int nums[5];  
int i, tot=0;  
for(i=0; i<5; i++) {  
    tot+=nums[i];  
}  
printf("total : %d\n", tot);
```



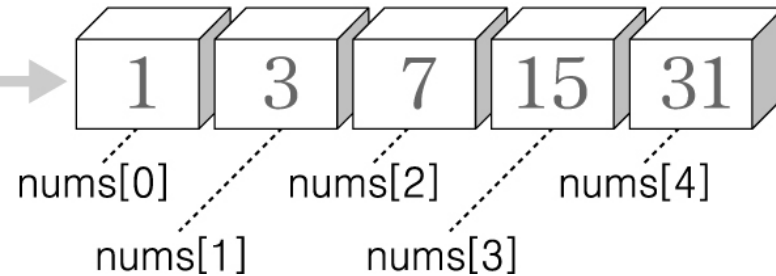
결과는 시스템의 상황에 따라 달라진다.

배열 초기화

- 배열은 기억공간의 수가 많으므로 중괄호를 사용하여 초기값을 나열

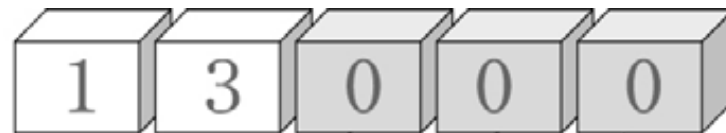
배열의 초기화 `int nums[5] = {1, 3, 7, 15, 31};`

순서대로 저장된다.



- 배열요소의 수보다 초기화 값이 적으면 남은 기억공간은 0으로 채워짐



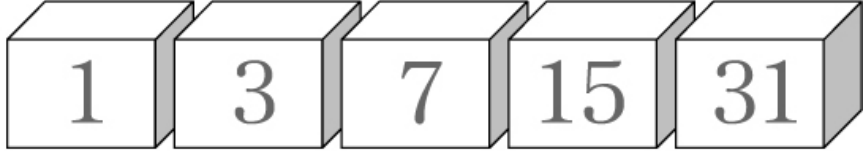
`int nums[5] = {1, 3};`



자동으로 0이 채워진다.

배열 초기화와 자동기능

- 배열을 선언할 때 초기화하면 배열요소의 개수를 생략할 수 있다.

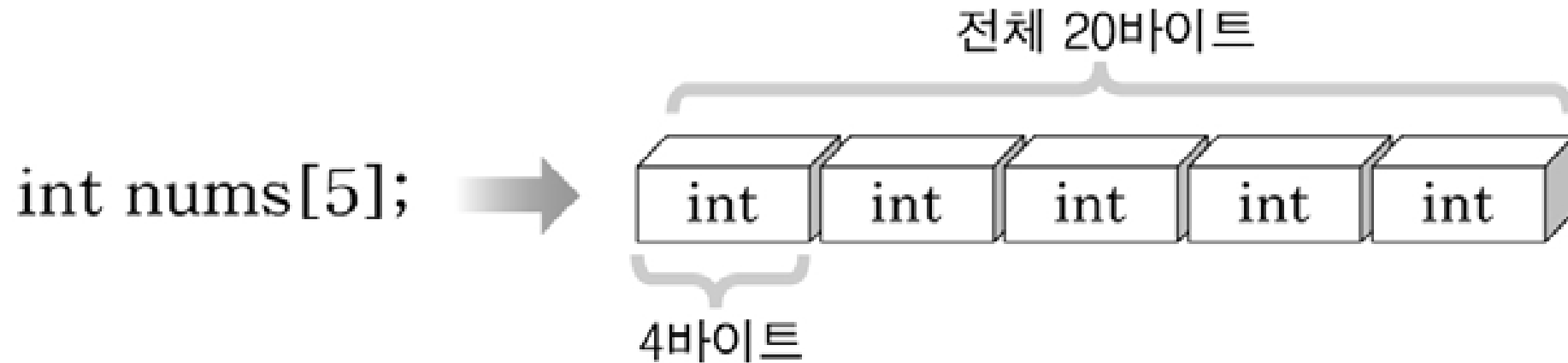
int nums  = {1, 3, 7, 15, 31};  

숫자 생략가능

다섯 개의 기억공간이 자동으로 잡힌다.

배열 크기 구하기

- 배열요소의 개수가 자동으로 계산되도록 프로그램을 작성하면 배열의 크기가 바뀌어도 프로그램을 수정할 필요가 없다.

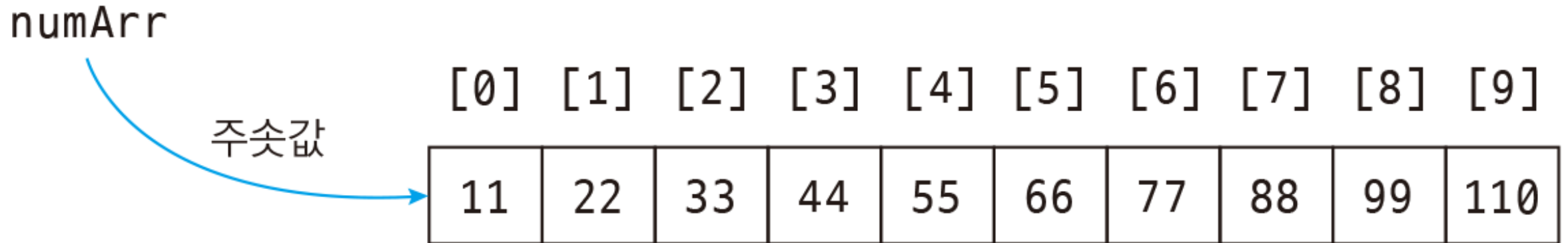


$$\text{배열요소의 개수} = \frac{\text{sizeof(nums)}}{\text{sizeof(nums[0])}}$$

배열 전체의 크기 배열요소 하나의 크기

배열을 포인터에 넣기

- 배열은 사실 첫 번째 요소의 주소값만 담고 있음



```
#include <stdio.h>

int main()
{
    int numArr[10] = { 11, 22, 33, 44, 55, 66, 77, 88, 99, 110 };
    int *numPtr = numArr;          // 포인터에 int형 배열을 할당
    printf("%d\n", *numPtr);
    printf("%d\n", *numArr);
    printf("%d\n", numPtr[5]);
    printf("%d\n", sizeof(numArr));
    printf("%d\n", sizeof(numPtr));
    return 0;
}
```

2차원 배열

자료형 배열이름 [세로크기] [가로크기];

자료형 배열이름 [세로크기] [가로크기] = {{값, 값, 값}, {값, 값, 값}};

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int numArr[3][4] = { // 세로 크기 3, 가로 크기 4인 int형 2차원 배열 선언
```

```
        { 11, 22, 33, 44 },
```

```
        { 55, 66, 77, 88 },
```

```
        { 99, 110, 121, 132 } 
```

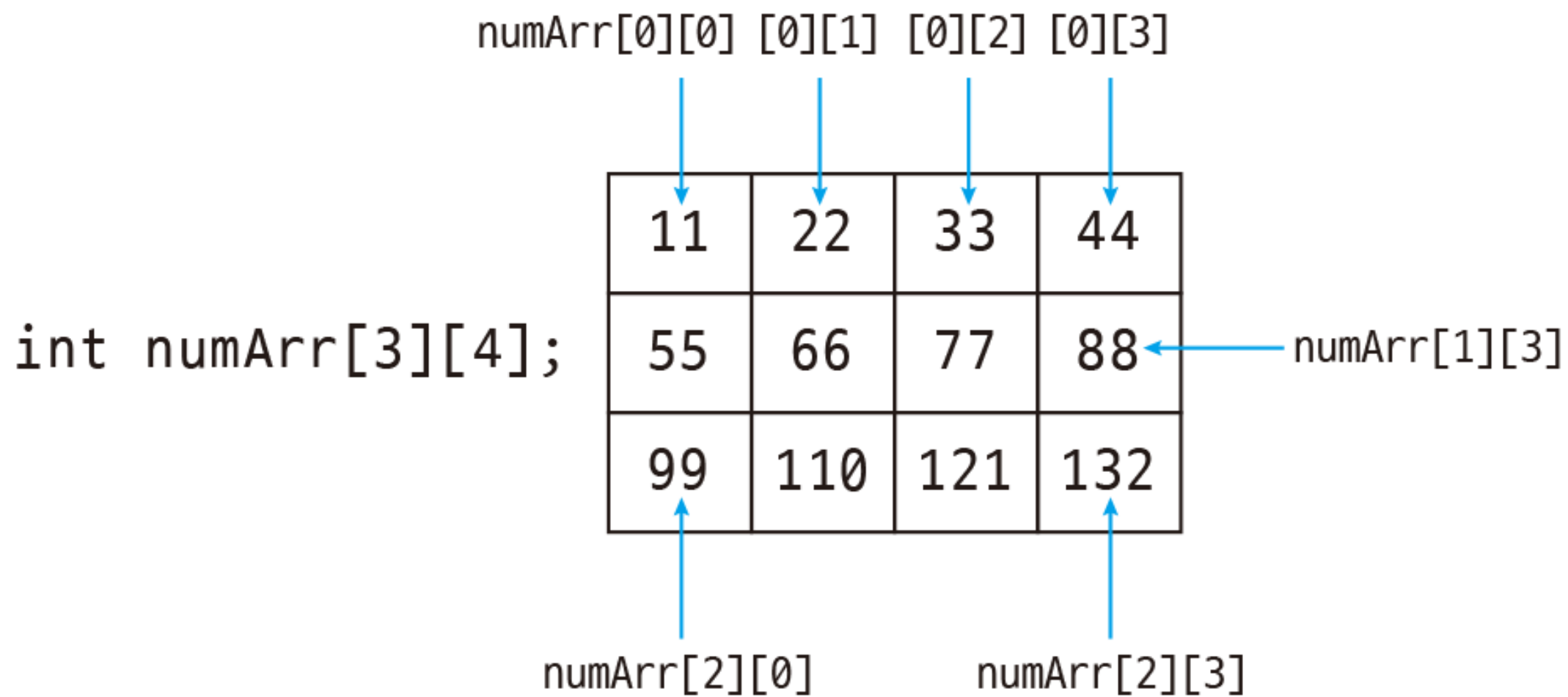
```
};
```

```
    printf("%d\n", numArr[0][0]);
```

```
    printf("%d\n", numArr[1][2]);
```

```
    return 0;
```

```
}
```



2차원 배열의 크기 구하기

```
#include <stdio.h>

int main()
{
    int numArr[3][4] = {
        { 11, 22, 33, 44 },
        { 55, 66, 77, 88 },
        { 99, 110, 121, 132 }
    };

    printf("%d\n", sizeof(numArr));
    int col = sizeof(numArr[0]) /
               sizeof(int);
    int row = sizeof(numArr) /
               sizeof(numArr[0]);
    printf("%d\n", col);
    printf("%d\n", row);
    return 0;
}
```

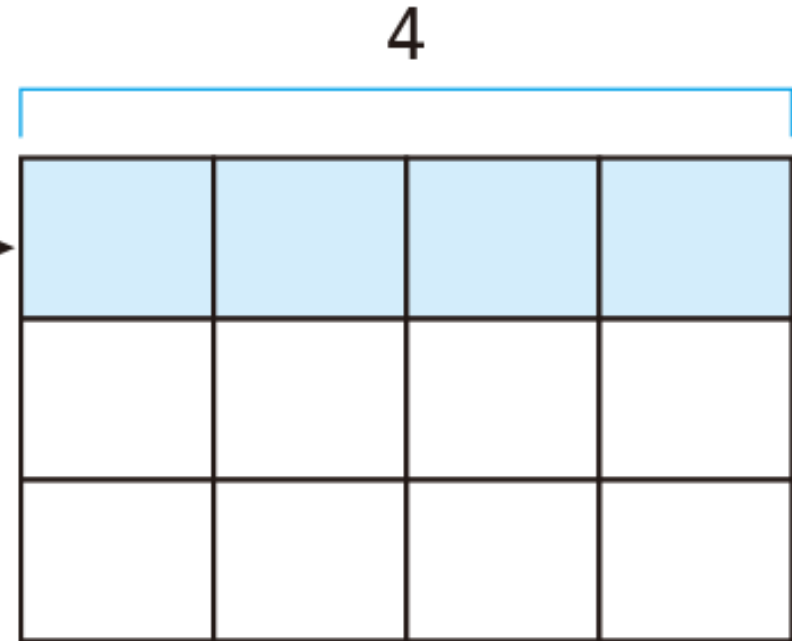

2차원 배열을 포인터에 넣기

자료형 (*포인터이름) [가로크기];

```
int (*numPtr)[4]
```



{ a, b, c, d }를 가리키는 포인터



```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int numArr[3][4] = {  
        { 11, 22, 33, 44 },  
        { 55, 66, 77, 88 },  
        { 99, 110, 121, 132 }  
    };
```

```
    int (*numPtr)[4] = numArr;
```

```
...
```